Classificação Automática de Processos em um Sistema Operacional de Uso Geral

Priscila V. Araujo[1], **Carlos A. Maziero**[2], **Julio C. Nievola**[1]

[1]PPGIa – Programa de Pós-Graduação em Informática
PUCPR – Pontifícia Universidade Católica do Paraná
Curitiba PR – Brasil

[2]DAInf – Departamento Acadêmico de Informática
UTFPR – Universidade Tecnológica Federal do Paraná
Curitiba PR – Brasil

{priscila.vriesman,nievola}@ppgia.pucpr.br, maziero@utfpr.edu.br

Resumo. *A tarefa mais importante do escalonador em um sistema operacional multitarefa de uso geral é propiciar uma fatia justa de tempo de processador a todos os processos, de forma a obter um bom desempenho e um tempo de resposta adequado às aplicações interativas. Cada processo tem demandas próprias de processamento e de tempo de resposta, que nem sempre podem ser facilmente informadas pelo usuário ou inferidas pelo escalonador. Este artigo tem por objetivo explorar as possibilidades de aplicação de técnicas de mineração de dados à massa de informações mantidas pelo núcleo do sistema para cada processo, visando 1) descobrir automaticamente grupos de processos com comportamento similar e 2) classificar automaticamente novos processos nesses grupos. A classificação automática dos processos em grupos de comportamento similar pode auxiliar significativamente a tarefa do escalonador de processos.*

Abstract. *The scheduler's main goal in a general purpose multitasking operating system is to provide a fair share of processor time to all processes, in order to achieve good performance and an adequate response time for interactive applications. Each process has its own demands for processing and response time, which can not always easily be informed by the user or inferred by the scheduler itself. This article aims to explore the possibilities of applying data mining techniques to the mass of information held by the system kernel for each process, in order to 1) automatically discover groups of processes with similar behavior and 2) automatically classify new processes in these groups. The automatic classification of processes into groups of similar behavior can significantly assist the task of the process scheduler.*

1. Introdução

O escalonamento de processos é o responsável por gerenciar de forma eficiente o controle de ordem e de tempo de execução dos processos em um sistema operacional, com a finalidade de satisfazer alguns objetivos conflitantes, como garantir que cada processo receba uma parte justa de processamento, minimizar o tempo de resposta para os usuários interativos e conciliar processos de alta prioridade com baixa prioridade.

Em um sistema operacional de uso geral típico, há uma grande diversidade de aplicações, além de programas auxiliares (geralmente associados à construção do ambiente do usuário) e serviços que executam em segundo plano, como serviços de rede e *daemons* diversos. Cada processo tem demandas próprias de processamento e de tempo de resposta, que nem sempre podem ser facilmente informadas pelo usuário ou inferidas pelo escalonador. Por isso, o provimento de informações adicionais sobre o comportamento dos processos ao escalonador pode auxiliar significativamente sua tarefa.

Este artigo tem por objetivo explorar as possibilidades de aplicação de técnicas de mineração de dados à massa de informações mantidas pelo núcleo do sistema para cada processo, visando 1) descobrir automaticamente grupos de processos com comportamento similar e 2) classificar automaticamente novos processos nesses grupos. Diversos algoritmos de classificação automática conhecidos são avaliados quanto à qualidade de seus resultados e ao seu custo de processamento.

O texto do artigo está dividido como segue: a Seção 2 relembra os principais conceitos sobre mineração de dados e suas técnicas; a Seção 3 descreve o problema da classificação de processos em sistemas operacionais; a Seção 4 apresenta os procedimentos utilizados para a construção do modelo de classificação; a Seção 5 apresenta os resultados obtidos, e por fim a conclusão é descrita na Seção 6.

2. Mineração de dados

Os sistemas computacionais têm como característica típica armazenar uma grande quantidade de informações. Contudo, muitas vezes o conhecimento relevante está literalmente escondido no meio das mesmas, sendo necessário explicitá-lo. A representação do processo de busca e extração do conhecimento é denominada *Descoberta de Conhecimento* (*Knowledge Discovery in Databases* – KDD), e envolve todo o ciclo que os dados percorrem até se transformarem em conhecimento. Para satisfazer esse objetivo, as etapas do chamado "processo KDD" são apresentadas em [Fayyad et al. 1996]:

1. *Seleção dos dados*: define um conjunto de dados pertencentes a um domínio, contendo todas as possíveis características e observações (registros);
2. *Pré-processamento e limpeza dos dados*: elimina ruídos no conjunto de dados extraído, removendo dados redundantes ou inconsistentes, como também recupera dados incompletos e avalia possíveis dados discrepantes (os chamados *outliers*);
3. *Transformação dos dados*: seleciona as características úteis para representar os dados, podendo reduzir a dimensionalidade do conjunto de dados, e realiza a formatação adequada dos dados a serem utilizados pelo algoritmo de aprendizagem;
4. *Mineração de dados*: analisa de forma automática ou semi-automática grandes bases de dados com objetivo de descobrir padrões, incluindo regras de classificação, regressão, agrupamento, modelagem de sequência e regras de associação;
5. *Interpretação e avaliação*: analisa os padrões descobertos, para que o conhecimento adquirido pela tarefa de mineração de dados seja interpretado e avaliado.

Dentre as etapas citadas, destaca-se a mineração de dados, que, de acordo com [Han and Kamber 2006], consiste no processo de explicitação do conhecimento com a utilização de técnicas e métodos que possibilitam realizar a análise de grandes quantidades de dados para a extração de conhecimento previamente desconhecido. O processo de

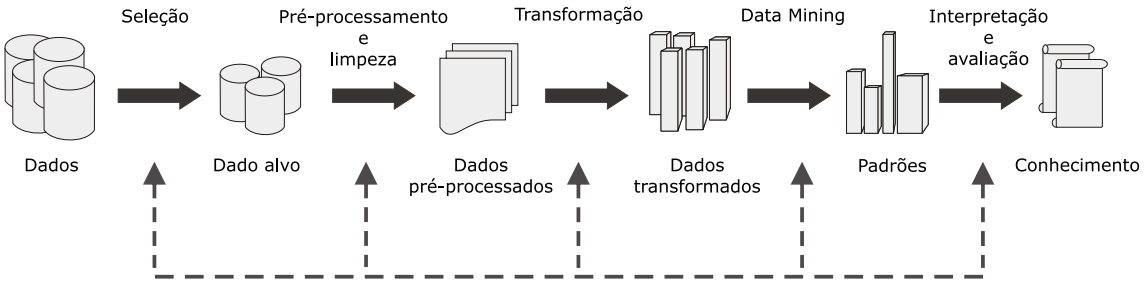

Figura 1. Etapas do processo KDD [Fayyad et al. 1996]

aprendizagem na mineração de dados funciona através da criação de modelos baseados em observações dos dados; há basicamente duas famílias de processos de aprendizagem:

- *Aprendizagem supervisionada*: fornece exemplos ao processo de aprendizagem, sendo que cada exemplo recebe uma classe associada que constitui o modelo que caracteriza uma classe.
- *Aprendizagem não-supervisionada*: baseia-se em observações e descobertas sem classe definida, e observa os exemplos a fim de reconhecer padrões.

Segundo esse enfoque, a mineração de dados é considerada uma técnica confiável com aplicação prática em diversas áreas. Ela consiste em técnicas e algoritmos específicos, permitindo produzir como resultado a construção de modelos computacionais, que se correlacionam com o objetivo proposto, e que buscam mais que a interpretação dos dados existentes, visando fundamentalmente generalizar possíveis fatos e correlações ainda não deduzidas nem facilmente percebidas.

3. Classificação de processos

Um problema que os escalonadores de processos tem de enfrentar é que cada processo é único e muitas vezes seu comportamento pouco previsível. Alguns processos são considerados como orientados a entrada/saída (*I/O bound*), pois gastam a maior parte de seu tempo em operações de entrada e saída de dados; já outros são orientados a processamento (*CPU bound*), pois necessitam de mais tempo de processador. Muitos processos podem alternar entre períodos de entrada/saída e de processamento. Tanto os processos *I/O bound* quanto *CPU bound* podem fazer parte também de diferentes classes de processos no que diz respeito às suas necessidades de escalonamento. Uma classificação usual dos processos nesse sentido consiste em separá-los em três grandes classes:

- *Batch*: processos que não necessitam de intervenção do usuário; têm como função tratar um conjunto de dados de entrada, realizar o processamento dos dados e produzir um conjunto de dados de saída, como programas de cálculo numérico, compilações e backups.
- *Interativos*: processos que interagem diretamente com o usuário, ou seja, são tarefas orientadas a entrada/saída geralmente através de uma interface gráfica, e que precisam de um tempo de resposta rápido do sistema operacional para o tratamento de requisições no contexto do usuário.
- *Daemons*: processos geralmente carregados durante a inicialização do sistema, que executam continuamente enquanto o sistema estiver ativo, e esperam em segundo plano até que algum serviço seja requerido. Um exemplo desta classe são

os processos de serviços de e-mail (como SMTP, POP e IMAP), pois tem a característica de estarem frequentemente ativos, mas se bloqueiam rapidamente.

A Tabela 1 descreve as principais características dessas classes.

Tabela 1. Características das classes de processos

Características	Batch	Daemon	Interativo
Interação com o usuário			•
Execução em segundo plano	•	•	
Alto custo computacional	•		
Baixo tempo de resposta			•

Como os vários tipos de processos de um sistema operacional tem características diferentes, eles devem ser tratados de forma distinta pelo escalonador, em função de suas demandas específicas. A maioria dos escalonadores de sistemas operacionais de uso geral privilegia os processos *I/O bound* em relação aos processos de *CPU bound*, de forma a melhorar o tempo de resposta das aplicações interativas. A adaptação do escalonador para certos tipos de aplicações, sobretudo as interativas, não é uma preocupação recente. Por exemplo, o trabalho [Nieh and Lam 1997] define um algoritmo de escalonamento hierárquico com características especiais para o tratamento de aplicações multimídia.

No núcleo Linux 2.6, a política de escalonamento foi desenhada com o objetivo de melhorar a interatividade, diminuindo o tempo de resposta das tarefas sensíveis à latência, através de um *estimador de interatividade* [Hussein et al. 2004, Torrey et al. 2007]. Ao mesmo tempo em que o escalonador realiza o escalonamento de processos, ele também estima a interatividade de cada aplicação, buscando determinar quais são as tarefas interativas e quais são as tarefas *CPU bound*. Além disso, o escalonador Linux possibilita o ajuste dinâmico das prioridades dos processos, de forma a acomodar mudanças no comportamento dos processos durante sua execução.

Alguns trabalhos realizados no objetivo de otimizar o escalonador de processos podem ser citados, como o estudo realizado por [Negi and Kishore 2005], que faz o uso de técnicas de aprendizagem de máquina para melhorar o desempenho do escalonador de processos no Linux, com a modificação do escalonador para minimizar o tempo de processamento na execução dos processos, que utilizou o algoritmo de árvore de decisão (algoritmo C4.5), mostrando sua efetividade na otimização do escalonador, e reduzindo o tempo de processamento na execução de processos.

Um outro trabalho na área é relatado por [Suranauwarat and Taniguchi 2001], que propôs um escalonador de processos que controla o compartilhamento de recursos na CPU observando os comportamentos dos processos, por meio de *logs* dos processos. Já o trabalho [Lim and Cho 2007] demonstra um método de escalonamento adaptativo, que busca extrair as características dos processos em tempo real, classificando os processos em classes e utilizando lógica nebulosa (*Fuzzy Logic*) para decidir a prioridade de cada processo. Essa abordagem, se comparada com os métodos de escalonamentos tradicionais, possui alguns benefícios, pois o escalonamento de processos é realizado de acordo com o tipo de processo (*batch*, *daemon*, *interativo*), provendo um método de escalonamento adaptativo de acordo com as preferências do usuário.

4. Geração do modelo de classificação

O objetivo deste trabalho consiste em explorar as possibilidades de aplicação de técnicas de mineração de dados à massa de informações mantidas pelo núcleo do sistema para cada processo, para descobrir grupos de processos com comportamento similar e classificar automaticamente novos processos nesses grupos. A classificação automática de processos em um sistema operacional pode ser considerada um problema complexo, tendo em vista a grande quantidade de processos com funções distintas e muitas características semelhantes entre eles. A construção do modelo de classificação de processos é alcançada através das seguintes atividades:

1. *Extração de atributos*: coletar informações dos processos presentes em um sistema Linux, gerando a base de treinamento;
2. *Agrupamento*: identificar os grupos existentes na massa de dados, através de um algoritmo de aprendizagem não-supervisionada;
3. *Análise de resultados*: analisar manualmente os grupos de processos obtidos na fase de agrupamento;
4. *Rotulação*: identificar a qual grupo pertence cada amostra e gerar uma base com os dados rotulados (denominada "base completa");
5. *Seleção de atributos*: reduzir a dimensionalidade do vetor de atributos através de algoritmos de seleção de atributos, gerando novas bases de treinamento;
6. *Classificação*: utilizar algoritmos de classificação para realizar o treinamento das amostras sobre as bases reduzidas;
7. *Geração do modelo*: avaliar o desempenho de cada algoritmo de classificação, em relação à taxa de acerto e ao tempo de processamento para o objetivo proposto.

Para não induzir resultados tendenciosos, parte-se do pressuposto que não há um conhecimento prévio sobre os grupos de processos existentes. Por isso, o tratamento dos dados foi realizado em duas etapas distintas: inicialmente uma etapa de aprendizagem não-supervisionada busca por indícios que possam relacionar processos entre si e construir um conjunto arbitrário de grupos; na sequência, um procedimento de aprendizagem supervisionada classifica os processos nos grupos previamente encontrados. No processo de criação do modelo, os métodos utilizados foram selecionados baseados no estudo da literatura da área correspondente, e por possuírem características de construção distintas. A Figura 2 demonstra o método utilizado para a construção do modelo de classificação.

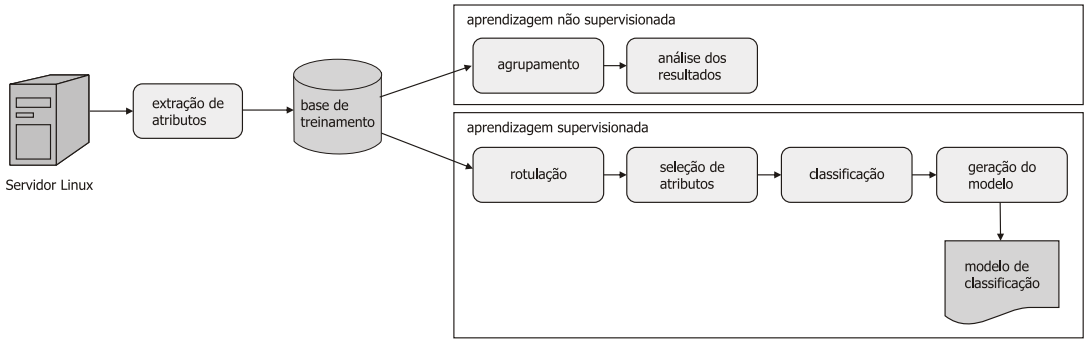

Figura 2. Geração do modelo de classificação

4.1. Extração de atributos

A fonte de dados para o experimento foi um servidor de terminais remotos SSH/VNC usado pelos alunos de graduação da PUCPR para seus trabalhos acadêmicos. O servidor executa a distribuição Linux Fedora Core 12, e tem como *hardware* um computador Sun V20Z com dois processadores AMD Opteron 248, 4GB de memória RAM, dois discos rígidos SCSI de 72 GB cada e duas interfaces de rede Gigabit Ethernet. O acesso ao mesmo se dá através de sessões remotas em modo texto (SSH) e gráfico (VNC), usando ambientes de desktop gráfico Gnome, OpenBox, LXDE e XFCE. O servidor possui cerca de 800 usuários cadastrados, mas a quantidade de usuários conectados simultaneamente é muito variável, observando-se com frequência mais de 10 usuários, e esporadicamente mais de 30 usuários simultâneos. Esse ambiente foi escolhido como fonte de dados pela riqueza e diversidade de processos que ele oferece: aplicações interativas, programas de cálculo intensivo, tratamentos em *batch*, um grande número de *daemons*, etc.

Na extração de atributos para a construção da base de dados, foram coletados dados dos processos do diretório virtual `/proc` do servidor Linux [Faulkner and Gomes 1991]. Nesse diretório virtual, cada subdiretório `/proc/`*`nnn`* contém arquivos com várias informações sobre o processo cujo PID é *`nnn`*. Os dados foram coletados por um programa desenvolvido em *Perl*, que extrai as informações dos processos, as formata em atributos e gera um arquivo em formato ARFF. Esse é o formato padrão de entrada da ferramenta WEKA [Bouckaert et al. 2002], usada nas demais etapas deste trabalho. Os dados foram coletados duas vezes por dia durante 185 dias, formando uma base de dados com 97.391 amostras distintas, com 108 atributos cada.

4.2. Agrupamento, análise e rotulação

Para descobrir padrões de processos nas amostras capturadas, foram realizados testes a partir de algoritmos de aprendizagem não-supervisionada, utilizando técnicas de agrupamento. Após diversos testes, foi selecionado o algoritmo de agrupamento *DBScan* [Ester et al. 1996]. Esse algoritmo tem como objetivo realizar agrupamentos baseado na densidade espacial, agrupando os objetos de forma arbitrária, separando os grupos de baixa densidade dos de alta densidade.

Através da aplicação do algoritmo de agrupamento *DBScan* foi possível identificar alguns grupos de processos, sendo observado que sempre os processos relativos a *threads* do núcleo do sistema operacional se mantém isolados, e que os demais tipos de processos, como *batch*, *daemons* e *interativos*, na maioria das vezes se encontravam visivelmente separados nos grupos identificados. A análise dos resultados obtidos através do agrupamento usando *DBScan* permitiu identificar 6 grupos distintos de processos, que receberam os seguintes rótulos em função de suas características e principais atribuições no contexto do sistema operacional:

- A (Aplicações interativas): todos os tipos de processos interativos (editores de texto, navegadores Web, clientes de e-mail, etc).
- D (*Daemons*): processos que executam em segundo plano e estão prontos para receber instruções.
- F (Funcionalidades de Desktop): processos que realizam tarefas de apoio ao ambiente de desktop gráfico (configuração, *framework* de componentes, etc).
- N (*Network*): processos envolvidos com a comunicação em rede.

- C (Comandos de texto): comandos simples de terminal em modo texto.
- K (*Kernel threads*): *threads* internas do núcleo do sistema operacional.
- O (outros): processos que não se enquadram nos demais grupos.

A Figura 3 mostra o gráfico de quantidade de amostras por rotulação.

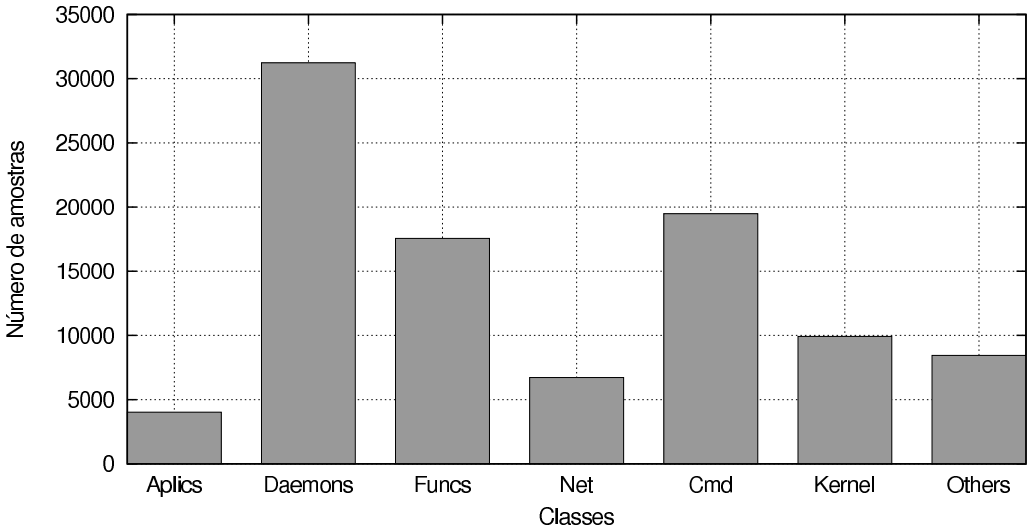

Figura 3. Quantidade de amostras por rotulação.

A partir dessa definição de grupos, a base de dados foi rotulada, ou seja, cada amostra da base foi acrescida de um atributo que representa a classe do respectivo processo, o que a torna preparada para a avaliação dos métodos de classificação automática.

4.3. Seleção de atributos

A seleção de atributos é uma etapa de pré-processamento que tem como objetivo descobrir quais atributos são os mais importantes, no sentido de descrever de maneira completa e não-redundante os dados da base [Liu and Yu 2005]. No processo de descoberta dos subconjuntos de atributos foi utilizada a abordagem *filtro*, que busca encontrar os atributos relevantes, ou seja, que apresentam alta correlação com as classes e baixa correlação com outros atributos, já que os atributos considerados irrelevantes podem atrapalhar a classificação, além de aumentar o custo computacional e o espaço de armazenamento.

De acordo com [Kohavi and John 1997], os algoritmos da abordagem *filtro* têm como objetivo selecionar um subconjunto de atributos que preserve as informações relevantes do conjunto inteiro de atributos. Os procedimentos utilizados para a busca do melhor subconjunto foram: *Busca Genética*, *Ranker*, *Ranker Search* e *Best-First*, e como procedimentos de avaliação foram escolhidos os métodos *Information Gain* e *CFS* [Liu and Yu 2005].

Os algoritmos de busca genética são baseados em mecanismos de seleção natural e genética, que adotam uma estratégia de busca paralela e estruturada, explorando as informações históricas para encontrar novos pontos de busca. Já os métodos de busca *Ranker* e *Rank Search* são baseados em estratégia de *ranking*, selecionando os primeiros melhores [Bouckaert et al. 2002]. Por fim, o método de busca *Best First* é baseado em pesquisa heurística, que tenta prever o melhor atributo candidato de acordo com regras

específicas. Contudo, apesar de possuírem os mesmos objetivos, esses métodos de seleção possuem implementações distintas e podem produzir subconjuntos de atributos distintos.

Assim, no intuito de facilitar o processo de classificação e avaliar o desempenho da classificação de acordo com a dimensionalidade dos atributos, foram criadas cinco novas bases de dados, formadas através da seleção de atributos. A Figura 4 mostra o processo de geração dos subconjuntos a partir da base de dados inicial (base completa), sobre a qual foram aplicados os métodos de seleção de atributos, gerando quatro bases de dados distintas (base BG, base RS, base RK, base BF). Por fim, também foi gerada uma base contendo os atributos selecionados por todos os métodos (base AC – *Atributos Comuns*). Os atributos que compõem cada base de dados estão descritos a seguir[1]:

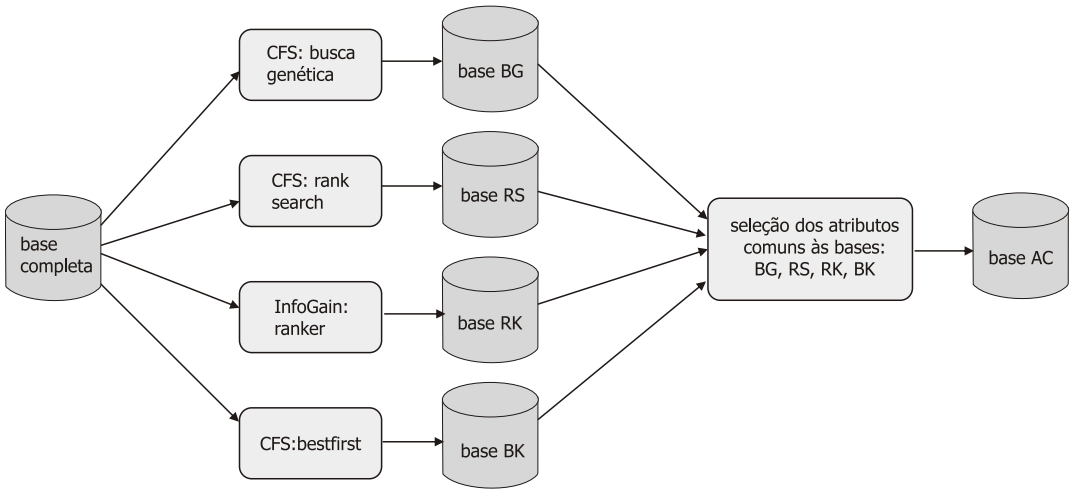

Figura 4. Processo de geração dos subconjuntos.

Base completa : formada pelos 108 atributos coletados: *fdinfo:openfiles, io:cancelled_write_bytes, io:rchar, io:read_bytes, io:syscr, io:syscw, io:wchar, io:write_bytes, maps:numlines, oom_score, sched:avg_atom, sched:avg_per_cpu, sched:nr_switches, sched:nr_voluntary_switches, sched:policy, sched:prio, sched:sched_info.bkl_count, sched:se.avg_overlap, sched:se.avg_running, sched:se.avg_wakeup, sched:se.block_max, sched:se.block_start, sched:se.exec_max, sched:se.exec_start, sched:se.iowait_count, sched:se.iowait_sum, sched:se.load.weight, sched:se.nr_failed_migrations_affine, sched:se.nr_failed_migrations_hot, sched:se.nr_failed_migrations_running, sched:se.nr_forced2_migrations, sched:se.nr_forced_migrations, sched:se.nr_involuntary_switches, sched:se.nr_migrations, sched:se.nr_wakeups, sched:se.nr_wakeups_affine, sched:se.nr_wakeups_affine_attempts, sched:se.nr_wakeups_local, sched:se.nr_wakeups_migrate, sched:se.nr_wakeups_remote, sched:se.nr_wakeups_sync, sched:se.sleep_max, sched:se.sleep_start, sched:se.slice_max, sched:se.sum_exec_runtime, sched:se.vruntime, sched:se.wait_count, sched:se.wait_max, sched:se.wait_start, sched:se.wait_sum, stat:blocked, stat:cmajflt, stat:cminflt, stat:comm, stat:cstime,*

[1]O nome de cada atributo indica sua localização; assim, o atributo *sched:se.nr_migrations* indica o valor do campo `se.nr_migrations` dentro do arquivo `/proc/PID/sched`. Contudo, alguns atributos representam valores agregados, como *fdinfo:openfiles*, que indica o número de entradas no diretório `/proc/PID/fdinfo`, ou seja, o número de descritores de arquivos abertos pelo processo. O significado de cada atributo pode ser consultado em [Hradílek et al. 2011].

stat:cutime, stat:delayacct_blkio_ticks, stat:endcode, stat:exit_signal, stat:flags, stat:kstkeip, stat:kstkesp, stat:majflt, stat:minflt, stat:nice, stat:num_threads, stat:pgrp, stat:pid, stat:policy, stat:ppid, stat:priority, stat:processor, stat:rss, stat:rsslim, stat:rt_priority, stat:session, stat:sigcatch, stat:sigignore, stat:signal, stat:startcode, stat:startstack, stat:starttime, stat:state, stat:stime, stat:tpgid, stat:tty_nr, stat:utime, stat:vsize, stat:wchan, statm:data, statm:resident, statm:share, statm:size, statm:text, status:FDSize, status:nonvoluntary_ctxt_switches, status:Tgid, status:TracerPid, status:VmData, status:VmExe, status:VmHWM, status:VmLib, status:VmPeak, status:VmPTE, status:VmRSS, status:VmSize, status:VmStk, status:voluntary_ctxt_switches.

Base BG : formada por 40 atributos selecionados pelo método *Busca Genética*, com avaliação por CFS: *fdinfo:openfiles, io:rchar, oom_score, sched:avg_atom, sched:prio, sched:se.iowait_sum, sched:se.load.weight, sched:se.nr_forced2_migrations, sched:se.nr_wakeups, sched:se.nr_wakeups_sync, sched:se.sleep_start, sched:se.wait_start, stat:cstime, stat:endcode, stat:flags, stat:num_threads, stat:pgrp, stat:pid, stat:priority, stat:rss, stat:rsslim, stat:rt_priority, stat:sigcatch, stat:sigignore, stat:startcode, stat:starttime, stat:state, stat:tpgid, stat:utime, stat:vsize, stat:wchan, statm:resident, statm:share, statm:size, statm:text, status:TracerPid, status:VmHWM, status:VmLib, status:VmPeak, status:VmStk.*

Base RS : formada por 19 atributos selecionados pelo método *Rank Search*, com avaliação por CFS: *sched:se.nr_failed_migrations_affine, stat:endcode, stat:flags, stat:kstkeip, stat:sigcatch, stat:sigignore, stat:startcode, stat:state, stat:tgpid, stat:tty_nr, stat:vsize, stat:wchan, statm:size, statm:text, status:TracerPid, status:VmExe, status:VmLib, status:VmPeak, status:VmSize.*

Base RK : formada por 10 atributos selecionados pelo método *Ranker*, com avaliação por *Information Gain*: *oom_score, stat:endcode, stat:vsize, statm:size, statm:text, status:VmData, status:VmExe, status:VmLib, status:VmPeak, status:VmSize.*

Base BF : formada por 9 atributos selecionados pelo método *Best First*, com avaliação por CFS: *stat:endcode, stat:ppid, stat:sigignore, stat:startcode, stat:tty_nr, stat:vsize, stat:wchan, statm:text, status:VmLib.*

Base AC : formada pelos 4 atributos que formam a interseção dos atributos das bases BG, RS, RK e BF: *stat:vsize, stat:endcode, statm:text, status:VmLib.*

Como a base AC representa os atributos presentes nos resultados de todos os algoritmos de seleção de atributos, pode-se presumir que estes representem as caracterícticas que melhor permitam diferenciar os vários tipos de processos e que são realmente importantes para o processo de classificação. O significado de cada um desses atributos é descrito a seguir :

- *stat:vsize*: memória virtual em uso pelo processo;
- *statm:text*: tamanho da memória usada pelo código do processo;
- *stat:endcode*: endereço abaixo do qual o código do processo pode executar;
- *status:VmLib*: espaço usado pelas bibliotecas compartilhadas do processo.

Observa-se que todos os atributos em comum estão relacionados ao uso de memória pelo processo, seja o tamanho total do processo, o tamanho de seu código ou de suas bibliotecas compartilhadas, ou ainda a região da memória em que o processo pode executar. Este é um resultado interessante, pois os estimadores de interatividade dos escalonadores em uso normalmente usam métricas baseadas no comportamento temporal dos processos e em sua demanda de processador [Hussein et al. 2004, Torrey et al. 2007].

4.4. Classificação

De acordo com [Han and Kamber 2006], a tarefa de classificação consiste na construção de um modelo com o objetivo de classificar dados ainda não conhecidos, visando categorizá-los numa classe pré-definida, baseando-se nas características comuns entre o conjunto de dados da base de dados de treinamento. Muitos algoritmos de classificação foram propostos na área de mineração de dados. Para aplicação no presente trabalho de classificação de processos foram selecionados os seguintes métodos, que são representativos dos principais paradigmas de classificação existentes:

- *C4.5*: cria uma árvore de decisão, na qual estão presentes somente os atributos considerados relevantes. Cada vez que um atributo é escolhido, os dados de treinamento são divididos em subgrupos, que correspondem aos diferentes valores dos atributos e o processo é repetido até que uma grande parte dos atributos pertençam a uma única classe.
- *OneR*: cria uma regra única para realizar a classificação com o menor percentual de erro possível. A regra é baseada na determinação de uma classe que possui o atributo mais frequente, podendo utilizar somente um atributo para a classificação.
- *Naive Bayes*: considerado um dos mais simples classificadores probabilísticos, é baseado na construção de um modelo de conjunto de probabilidades. As probabilidades são estimadas pela frequência de cada valor de característica para as instâncias dos dados de treinamento.
- *MLP*: as redes neurais *Perceptron Multi-Camadas* (MLP), se baseiam na representação de conhecimento sob a forma de pesos nas conexões entre suas unidades fundamentais, os neurônios, sendo os padrões de treinamento apresentados à entrada e mapeados num conjunto de padrões de saída da rede.

5. Resultados e discussões

Os métodos de classificação automática descritos na seção anterior (*C4.5*, *OneR*, *Naive Bayes* e *MLP*) foram aplicados às seis bases de dados de treinamento descritas na Seção 4.3. Para evitar a necessidade de gerar novos dados para avaliar os métodos de classificação, foi adotada a técnica de *hold-out*, que consiste em dividir a base original em duas partes: uma parte para treinamento, ou seja, construção do modelo, composta por aproximadamente 2/3 dos dados originais e outra parte, com o 1/3 restante, utilizada para avaliar a qualidade do modelo gerado com os dados de treinamento.

Os resultados mensurados foram: 1) percentagem de acerto (ou número de instâncias corretamente classificadas); 2) tempo para a construção do modelo (ou tempo de aprendizagem). As Tabelas 2 e 3 apresentam os resultados obtidos, em função do número de atributos em cada base. Constatou-se uma forte correlação entre o tempo de processamento e o número de atributos da base utilizada pelos algoritmos de classificação. Essa correlação também existe entre a taxas de acerto e o numero de atributos, embora seja menos intensa. Os algoritmos *C4.5* e *OneR* apresentaram um ótimo desempenho, acima de 95% em todos os testes, sendo o primeiro ligeiramente melhor que o segundo. Contudo, o algoritmo *OneR* tem um custo computacional uma ordem de grandeza inferior ao *C4.5*. Ambos os algoritmos tiveram resultados satisfatórios com a base AC, que contém o menor número de atributos (e portanto o menor custo computacional).

Tabela 2. Taxa de acerto dos algoritmos de classificação com *hold-out*

Bases	Atributos	C4.5	MLP	OneR	Naive Bayes
Base AC	4	97,31%	80,47%	95,83%	51,44%
Base BF	9	98,64%	85,81%	95,85%	51,01%
Base RK	10	97,23%	87,16%	95,85%	53,29%
Base RS	19	98,78%	83,93%	95,83%	54,97%
Base BG	40	99,86%	92,45%	96,10%	59,38%
Base completa	108	99,76%	91,32%	95,85%	58,27%

Tabela 3. Tempo de execução dos algoritmos de classificação com *hold-out*

Bases	Atributos	C4.5	MLP	OneR	Naive Bayes
Base AC	4	4,70s	158,88s	0,27s	0,35s
Base BF	9	8,23s	305,71s	0,58s	0,62s
Base RK	10	8,98s	317,13s	1,03s	0,84s
Base RS	19	20,42s	886,00s	1,56s	1,52s
Base BG	40	70,17s	2.280,56s	6,05s	2,93s
Base completa	108	120,24s	7.593,89s	11,48s	8,35s

A principal diferença entre os algoritmos *OneR* e *C4.5* é que o primeiro fornece apenas uma única regra de classificação, usando um único atributo para fornecer a classificação, enquanto o segundo fornece várias regras, podendo ser mais útil no sentido de apresentar as condições para a classificação em cada uma das classes encontradas. A escolha do melhor algoritmo irá portanto depender do custo computacional que pode ser aceito, bem como da taxa de acerto exigida.

Quanto aos demais algoritmos de classificação, observou-se que o algoritmo *Naive Bayes* se torna inviável para a classificação de processos, pois não apresenta bons resultados. Já o algoritmo MLP possui um percentual de acerto aceitável, mas seu tempo de processamento para a criação do modelo pode ser considerado excessivamente alto.

6. Conclusão

Este artigo traz uma abordagem comparativa entre diferentes algoritmos de seleção de atributos e classificação, com o objetivo de identificar o melhor modelo para a classificação automática de processos em um sistema operacional usando como atributos as informações providas pelo sistemas de arquivos virtual `/proc`.

Considerada uma das mais importantes tarefas em mineração de dados, a classificação tem seu desempenho diretamente afetado por atributos que sejam redundantes ou irrelevantes. Observando os algoritmos utilizados, (*C4.5*, *MLP*, *Naive Bayes* e *OneR*), pode-se concluir que os algoritmos *C4.5* e *OneR* apresentam os melhores resultados em relação ao percentual de acerto e ao tempo de treinamento; além disso, não é necessária a utilização dos 108 atributos inicialmente definidos para uma boa classificação. Sendo assim, utilizando somente os 4 atributos escolhidos por todos os métodos de seleção (*stat:vsize*, *statm:text*, *stat:endcode* e *status:VmLib*), pode-se obter uma taxa de acerto superior a 95% com um custo computacional bastante baixo, o que é importante para permitir sua execução frequente.

Como continuação para o presente trabalho, pretende-se investigar de que forma os grupos de processos definidos pelo método de agrupamento podem ser mapeados no conceito de grupos de processos definido pelo escalonador CFS (*Completely Fair Scheduler*) das versões mais recentes do núcleo Linux. Outro problema relevante a considerar é a possibilidade de mudança de comportamento de um processo, o que implicaria na reclassificação do mesmo (ou em sua pertinência a mais de uma classe de processos).

Referências

Bouckaert, R., Frank, E., Kirkby, R., Reutemann, P., Seewald, A., and Scuse, D. (2002). *WEKA Manual for Version 3.7.2*. University of Waikato, New Zealand.

Ester, M., Kriegel, H. P., X., J. S., and Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data mining*, volume 1996, pages 226–231. AAAI Press.

Faulkner, R. and Gomes, R. (1991). The process file system and process model in UNIX System V. In *USENIX Conference Proceedings*.

Fayyad, U., P., G. P.-S., and Smyth (1996). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34.

Han, J. and Kamber, M. (2006). *Data mining: concepts and techniques*. Morgan Kaufmann.

Hradílek, J., Silas, D., Prpič, M., Kopalová, E., Ella, Lackey, D., Ha, J., O'Brien, D., Hideo, M., and Domingo, D. (2011). *Red Hat Enterprise Linux 6 – Deployment Guide*. Red Hat Inc. Appendix C – The `proc` File System.

Hussein, N., Kolivas, C., Haron, F., and Yong, C. (2004). Extending the Linux operating system for grid computing. In *APAN Network Research Workshop*.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324.

Lim, S. and Cho, S. B. (2007). Intelligent OS process scheduling using fuzzy inference with user models. *New Trends in Applied Artificial Intelligence*, pages 725–734.

Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502.

Negi, A. and Kishore, K. P. (2005). Applying machine learning techniques to improve Linux process scheduling. In *TENCON IEEE Region 10*, pages 1–6. IEEE.

Nieh, J. and Lam, M. (1997). The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *ACM Symposium on Operating Systems Principles*, pages 506–519.

Suranauwarat, S. and Taniguchi, H. (2001). The design, implementation and initial evaluation of an advanced knowledge-based process scheduler. *ACM SIGOPS Operating Systems Review*, 35(4):61–81.

Torrey, L. A., Coleman, J., and Miller, B. P. (2007). A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. *Software: Practice and Experience*, 37(4):347–364.